**Yee &
Associates, P.C.**

4100 Alpha Road
Suite 1100
Dallas, Texas 75244

Main No. (972) 385-8777
Facsimile (972) 385-7766

Facsimile Cover Sheet

To: Commissioner for Patents for **Examiner Satish Rampuria** **Group Art Unit 2191**	Facsimile No.: **571/273-8300**
From: Stephanie Fay Legal Assistant to Betty Formby	No. of Pages Including Cover Sheet: **31**

Message:

Enclosed herewith:

- Transmittal of Appeal Brief;
- Appeal Brief; and
- Evidence cited in Appeal Brief.

Re: Application No. **10/041,127**
Attorney Docket No: **RSW920010133US1**

Date: Friday, March 03, 2006

Please contact us at (972) 385-8777 if you do not receive all pages indicated above or experience any difficulty in receiving this facsimile.	*This Facsimile is intended only for the use of the addressee and, if the addressee is a client or their agent, contains privileged and confidential information. If you are not the intended recipient of this facsimile, you have received this facsimile inadvertently and in error. Any review, dissemination, distribution, or copying is strictly prohibited. If you received this facsimile in error, please notify us by telephone and return the facsimile to us immediately.*

PLEASE CONFIRM RECEIPT OF THIS TRANSMISSION BY FAXING A CONFIRMATION TO 972-385-7766.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Hogstrom et al.** § Group Art Unit: **2191**

Serial No.: **10/041,127** § Examiner: **Rampuria, Satish**

Filed: **January 8, 2002** § Attorney Docket No.: **RSW920010133US1**

For: **Method, Apparatus, and Program to Determine the Mutability of an Object at Loading Time**

36736
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Certificate of Transmission Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being transmitted via facsimile to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, facsimile number (571) 273-8300 on March 3, 2006.

By: Stephanie Fay

TRANSMITTAL OF APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:
ENCLOSED HEREWITH:

- Appeal Brief (37 C.F.R. 41.37); and
- Evidence cited in Appeal Brief.

A fee of $500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0461. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0461. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0461.

Respectfully submitted,

Betty Formby
Registration No. 36,536
AGENT FOR APPLICANTS

Duke W. Yee
Registration No. 34,285
ATTORNEY FOR APPLICANTS

YEE & ASSOCIATES, P.C.
P.O. Box 802333
Dallas, Texas 75380
(972) 385-8777

Docket No. RSW920010133US1

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Hogstrom et al.** §
§ Group Art Unit: **2191**
Serial No. **10/041,127** §
§ Examiner: **Rampuria, Satish**
Filed: **January 8, 2002** §
§
For: **Method, Apparatus, and Program to Determine the Mutability of an Object at Loading Time** §
§

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Certificate of Transmission Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being transmitted via facsimile to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, facsimile number (571) 273-8300 on March 3, 2006.

By: Stephanie Fay
Stephanie Fay

APPEAL BRIEF (37 C.F.R. 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on January 3, 2006.

A fee of $500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0461. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0461. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0461.

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-22

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: 6-7, 10, 16-17, and 20
2. Claims withdrawn from consideration but not canceled: None
3. Claims pending: 1-5, 8-9, 11-15, 18-19, and 21-22
4. Claims allowed: None
5. Claims rejected: 1-5, 8-9, 11-15, 18-19, and 21-22
6. Claims objected to: None

C. CLAIMS ON APPEAL

The claims on appeal are: 1-5, 8-9, 11-15, 18-19, and 21-22

STATUS OF AMENDMENTS

No amendments have been filed since the issuance of the final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

A. CLAIM 1 - INDEPENDENT

The subject matter of claim 1 is directed to a method in a data processing system for loading a class (shown in **Figures 7** and **8**, discussed on page 20, line 19 through page 21, line 26). The method contains the following steps:

- loading a class (start block in **Figure 7**, page 20, lines 22-23);
- inserting a flag that will mark the class as immutable or not (step **702**, page 20, lines 22-23);
- determining whether or not the class is immutable (step **708**, page 20, lines 27-28);
- setting the immutability flag if the class is immutable (step **710**, page 20, lines 28-29);
- receiving a request to invoke a server application (start block in **Figure 8**, page 21, lines 6-8);
- examining an argument in the request (step **802**, page 21, lines 6-8);
- if the argument is an object ("yes" to step **804**, page 21, lines 17-19), identifying the class that describes the object (not specifically shown);
- determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set (step **812**, page 21, lines 17-19);
- if the object is immutable ("yes" to step **812**), passing an argument to the server application that includes a reference to the object, the argument not including a copy of the object (step **816**, page 21, lines 23-26); and
- if the object is mutable ("no" to step **812**), passing a copy of the object to the server application (step **814**, page 21, lines 19-21).

B. CLAIM 8 – INDEPENDENT

The subject matter of claim 8 is directed to a method in a data processing system for invoking an object (**Figure 8**, page 19, line 9 - page 20, line 7). The method contains the following steps:

- receiving, from a caller, a request to invoke an object (start block in **Figure 8**, page 21, lines 6-8);
- examining an argument in the request (step **802**, page 21, lines 6-8);
- if the argument is an object ("yes" to step **804**, page 21, lines 17-19), determining whether the object is immutable (step **812**, page 21, lines 17-19);
- if the object is immutable ("yes" to step **812**), passing an argument to the caller that includes a reference to the object and does not include a copy of the object (step **816**, page 21, lines 23-26); and
- if the object is mutable ("no" to step **812**), passing a copy of the object to the caller (step **814**, page 21, lines 19-21).

C. CLAIM 11 - INDEPENDENT

The subject matter of claim 1 is directed to an apparatus for loading a class (**Figures 3-5**, page 9, line 24 through page 17, line 25). The apparatus has the following elements:

- loading means for loading a class (**510** of **Figure 5**, page 15, lines 11-14);
- insertion means for inserting an immutability flag into the class (**404** of **Figure 4**, page 12, lines 7-24);
- first determination means for determining whether the class is immutable (**404** of **Figure 4**, page 12, lines 7-24);
- setting means for setting the immutability flag if the class is immutable (**404** of **Figure 4**, page 12, lines 7-24);
- a request to invoke a server application, the request including an argument, the request being an object (**Figures 6A-6B**, 1., ARG-#2, page 18, line 9 through page 20, line 8);
- a class that describes the object (not specifically shown);
- determining means for determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set (**404** of **Figure 4**, page 12, lines 7-24);

- if the object is immutable, an argument that is passed to the server application that includes a reference to the object and does not include a copy of the object (**Figure 6C**, ARG-#2 = reference, page 20, lines 8-18); and
- if the object is mutable, a copy of the object being passed to the server application object (**Figure 6B**, ARG-#2 = object', page 19, line 26 through page 20, line 7).

D. CLAIM 12 - DEPENDENT

The subject matter of claim 12 is directed to further defining the first determination means of claim 11 to include parsing means for parsing the bytecode of the class (**404** of **Figure 4**, page 12, lines 7-24).

E. CLAIM 13 - DEPENDENT

The subject matter of claim 13 is directed to further defining the first determination means of claim 11 as containing second determination means for determining whether the class can be modified after it is created (**404** of **Figure 4**, page 12, lines 7-24).

F. CLAIM 14 - DEPENDENT

The subject matter of claim 14 is directed to further defining the second determination means of claim 13 as containing means for determining whether all properties of the class are marked private (**404** of **Figure 4**, page 12, lines 7-24).

G. CLAIM 15 - DEPENDENT

The subject matter of claim 15 is directed to further defining the second determination means of claim 13 as containing means for determining whether there are any non-private methods that update properties of the class (**404** of **Figure 4**, page 12, lines 7-24).

H. CLAIM 18 – INDEPENDENT

The subject matter of claim 18 is directed to an apparatus for invoking an object (**Figures 3-5**, page 9, line 24 through page 17, line 25). The apparatus contains the following elements:

- receipt means for receiving, from a caller, a request to invoke an object (**404** of **Figure 4**, page 12, lines 7-24);
- examination means for examining an argument in the request (**404** of **Figure 4**, page 12, lines 7-24);
- determination means for determining whether the object is immutable if the argument is an object; (**404** of **Figure 4**, page 12, lines 7-24)
- reference means for passing an argument to the caller that includes a reference to the object and does not include a copy of the object (**404** of **Figure 4**, page 12, lines 7-24); and
- if the object is mutable, the reference means for passing a copy of the object to the caller (**404** of **Figure 4**, page 12, lines 7-24).

I. CLAIM 11 - DEPENDENT

The subject matter of claim 19 is directed to further defining the determination means of claim 18 as containing means for determining whether an immutability flag that is inserted into a class that describes the object is set (**404** of **Figure 4**, page 12, lines 7-24).

J. CLAIM 21 - INDEPENDENT

The subject matter of claim 21 is directed to a computer program product, for loading a class (not specifically shown, page 22, lines 6-18). Claim 21 is substantially a computer program product version of method claim 1.

K. CLAIM 22 - INDEPENDENT

The subject matter of claim 22 is directed to a computer program product for invoking an object (not specifically shown, page 22, lines 6-18). This claim is substantially a computer program product version of claim 8.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. GROUND OF REJECTION 1 (Claims 1-5, 8-9, 11-15, 18-19, and 21-22)

Claims 1-5, 8-9, 11-15, 18-19, and 21-22 stand rejected under 35 U.S.C. § 102(e) as anticipated over **Santosuosso** et al., Preventing Garbage Collection of Objects in Object Oriented Computer Programming Languages, U.S. Patent No.6,701,520, March 2, 2004 (hereinafter "**Santosuosso**").

ARGUMENT

A. GROUND OF REJECTION 1 (Claims 1-5, 8-9, 11-15, 18-19, and 21-22)

Claim 1 is representative of this group of claims and recites:

> 1. A method in a data processing system for loading a class, the method comprising:
> loading a class;
> inserting an immutability flag into the class;
> determining whether the class is immutable;
> setting the immutability flag if the class is immutable;
> receiving a request to invoke a server application;
> examining an argument in the request;
> if the argument is an object, identifying the class that describes the object;
> determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set;
> if the object is immutable, passing an argument to the server application that includes a reference to the object, the argument not including a copy of the object; and
> if the object is mutable, passing a copy of the object to the server application.

In rejecting this claim, the final office action states:

> Santosuosso disclose[s]: ...
> - determining whether the class is immutable (col. 5, lines 33-34 "Java compiler identifies any immutable objects");
> - setting the immutability flag if the class is immutable (col. 5, lines 33-36 "Java compiler identifies any immutable objects... and changes or "elevates" them to equivalent static class variables") ...

Final Office Action of November 16, 2005, item 8, page 4.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983).

The invention recited in claim 1, is not anticipated by **Santuosso** for at least two reasons. First, the cited section of **Santosuosso** is not checking a property of a class, but the property of an object. Further, **Santosuosso** does not actually determine mutability (or immutability), as this property would be understood by one of ordinary skill in the art, and therefore any flag set does not indicate immutability as this term is commonly understood.

Santosuosso Doesn't Check Class

In the rejection quoted above, a phrase from **Santosuosso** is cited as reading on the determining step. The entire sentence associated with that phrase states: "*In the invention described herein, the Java compiler identifies any immutable objects created with constants at compilation time and changes or "elevates" them to equivalent static class variables, resolving any name conflicts*". In this excerpt, **Santosuosso** identifies objects created with constants; this patent does not determine the mutability of a class. An object is a member of a class, but an object is not the same as a class. Therefore, the claimed feature of "*determining whether the class is immutable*" is not met by **Santosuosso** and claim 1 is not anticipated.

Santosuosso Doesn't Determine Immutability

Additionally, Appellants submit that although one can find quotes in **Santosuosso** asserting a check for immutability, this patent does not use the term *immutability* in the same manner as that term is used in the present application. Neither does **Santosuosso** use immutability in the same manner as it is used by one of ordinary skill in the art.

The present application defines the term *immutable* on the second page of the specification, which states, "*There are instances, however, when a passed object does not need to be copied because it is immutable, meaning that once the object is created it cannot be modified*" (Application, page 2, lines 14-17). Appellant's usage is consistent with both the definition in a general-purpose dictionary, such as Merriam-Webster, and with the common usage in object oriented language. The Evidence Appendix at the end of this Appeal Brief contains three separate documents defining the term *immutable* or *immutable object*: Merriam-Webster Online (m-w.com), JavaPractices.com, and ChurchillObjects.com. Merriam-Webster defines *immutable* as "*not capable of or susceptible to change*"; JavaPractices.com defines immutable objects as "*simply objects whose state (the object's data) does not change after construction*"; and

ChurchillObjects.com states that immutable "*simply means unchangeable in object-oriented parlance*".

In contrast, one of ordinary skill in the art would understand, upon a close examination of **Santosuosso**, that this patent is actually locating and managing a much narrower group of objects, i.e., constant objects and/or those immutable objects that will never need to have more than one instance of the object exist at a time. Thus, **Santosuosso** uses the term *immutable* in a manner that is narrower than both usage in the present application and common usage. **Santosuosso** is not determining immutability, as that term is generally understood, nor does the associated flag reflect the common understanding of *immutable*. Support for this assertion can be seen in **Santosuosso** and the reasons for this assertion are given in detail below.

First, it is noted that the examples given in **Santosuosso** use constants for the *immutable* objects. Constants are immutable, but they are not the only immutable objects and a determination whether an object is a constant is not the same as a determination of whether the object is immutable.

In a first example in column 5, this patent uses the illustration of the constant String ("Hello"), and notes:

> According to principles of the invention applied as described herein, immutable objects created with **constants**, herein referred to as fixed, in Java methods are treated as class variables which allows these objects to be created only once. Thus, if a method which creates an immutable object having a constant is called multiple times or if separate threads call the method, the object is created only once no matter how many times the method is invoked.

Santosuosso, column 5, lines 15-23, emphasis added.

In a second example, the patent points to Figure 1 (reproduced below) and notes:

> Thus, by way of example only, root class 140 contains the code which instantiates each of the constant objects: s=new String ("Hello"); ii=Integer (5); bb=Boolean (true); cc=BigInteger (10). Reassignment of constant objects can be done for all the aforementioned objects so long as the object is not passed to a JNI routine that could potentially modify the value.

Santosuosso, column 5, lines 52-58, emphasis added.

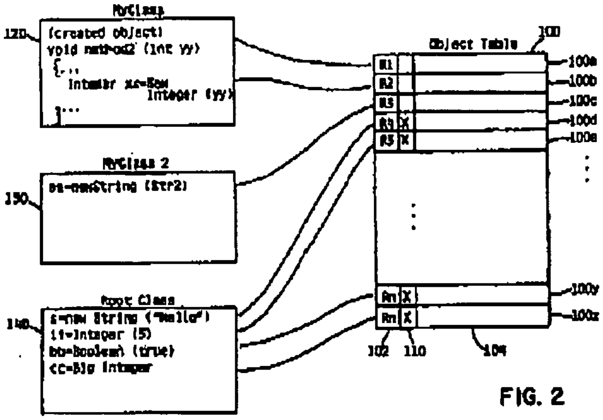

FIG. 2

In both of these examples, **Santosuosso** explicitly uses constants, not merely immutable objects. In addition to the examples, which by themselves are only illustrations, **Santosuosso** makes the following statements:

> In the invention described herein, the Java compiler identifies any immutable objects created with constants at compilation time and changes or "elevates" them to equivalent static class variables, resolving any name conflicts. Immutable or constant objects are objects whose internal value can be set only at creation time and a static class variable is one in which only one copy exists for all objects of that class. The act of elevation can be extended to other objects that are private and mutable object variables wherein the compiler determines that the values of the variables cannot be modified at run-time. A private object is one that can only be accessed by the class in which it is contained. These private mutable object variables, for purposes of this invention, are also considered fixed.

Santosuosso, column 5, lines 33-46, emphasis added.

As this excerpt shows, **Santosuosso** is identifying "*immutable objects created with constants at compilation time*", not merely immutable objects.

In addition to the direct evidence cited above, the specification of **Santosuosso** provides further indirect evidence of this patent's use of the word immutable. By raising the object to a static class variable, **Santosuosso** ensures that only one instance of the object exists, no matter how many applications are using the object at the same time (see both excerpts above). The use of only one instance works well if the object is a constant, i.e., the value of the object is same every time the object is instantiated. However, an object A can be immutable and yet have different values in each instantiation, e.g., the value of the object can be set at creation to an input value received from a user or from other code. If **Santosuosso** elevated object **A**, which in this example is immutable but not a constant, then **Santosuosso** could not allow multiple threads to utilize object **A** at the same time. To achieve this, **Santosuosso** would need to ensure that each thread that utilized object **A** was completely finished with object **A** before allowing another thread to utilize the object. Otherwise, with only one copy of object **A**, the value of **A** could be changed by each new thread when the thread instantiated object **A**, causing erroneous results.

Appellants have provided evidence from **Santosuosso** that this patent is using the term "immutable" differently from the manner in which it is used in the present application and also differently from the manner used by one of ordinary skill in the art, thus appearing, in a casual reading, to manage all immutable objects while in fact only working for objects that are constants or otherwise have much narrower properties than just "immutable". **Santosuosso** provides both direct and indirect evidence that "immutable" is too broad a term to accurately apply to the objects that are "elevated" in Santosuosso.

Thus, contrary to the assertion in the rejection, **Santosuosso** does not meet the steps of "determining whether the class is immutable; [and] setting the immutability flag if the class is immutable", as recited in claim **1**. The rejection should therefore be overturned.

The Board of Appeals is respectfully requested to overturn the outstanding rejection of these claims and indicate this application to be allowable.

Betty Formby

Betty Formby
Reg. No. 36,536
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

CLAIMS APPENDIX

The text of the claims involved in the appeal are:

1. A method in a data processing system for loading a class, the method comprising:

loading a class;

inserting an immutability flag into the class;

determining whether the class is immutable;

setting the immutability flag if the class is immutable;

receiving a request to invoke a server application;

examining an argument in the request;

if the argument is an object, identifying the class that describes the object;

determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set;

if the object is immutable, passing an argument to the server application that includes a reference to the object, the argument not including a copy of the object; and

if the object is mutable, passing a copy of the object to the server application.

2. The method of claim 1, wherein the step of determining whether the class is immutable comprises:

parsing the bytecode of the class.

3. The method of claim 2, wherein the step of determining whether the class is immutable further comprises:

determining whether the class can be modified after it is created.

4. The method of claim 3, wherein the step of determining whether the class can be modified comprises determining whether all properties of the class are marked private.

5. The method of claim 3, wherein the step of determining whether the class can be modified comprises determining whether there are any non-private methods that update properties of the class.

8. A method in a data processing system for invoking an object, the method comprising:

receiving, from a caller, a request to invoke an object;

examining an argument in the request;

if the argument is an object, determining whether the object is immutable;

if the object is immutable, passing an argument to the caller that includes a reference to the object and does not include a copy of the object; and

if the object is mutable, passing a copy of the object to the caller.

9. The method of claim 8, wherein the step of determining whether the object is immutable comprises:

determining whether an immutability flag that is inserted into a class that describes the object is set.

11. An apparatus for loading a class, comprising:

loading means for loading a class;

insertion means for inserting an immutability flag into the class;

first determination means for determining whether the class is immutable;

setting means for setting the immutability flag if the class is immutable;

a request to invoke a server application, the request including an argument, the request being an object;

a class that describes the object;

determining means for determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set;

if the object is immutable, an argument that is passed to the server application that includes a reference to the object and does not include a copy of the object; and

if the object is mutable, a copy of the object being passed to the server application.

12. The apparatus of claim 11, wherein the first determination means comprises:

parsing means for parsing the bytecode of the class.

13. The apparatus of claim 12, wherein the first determination means further comprises:

second determination means for determining whether the class can be modified after it is created.

14. The apparatus of claim 13, wherein the second determination means comprises means for determining whether all properties of the class are marked private.

15. The apparatus of claim 13, wherein the second determination means comprises means for determining whether there are any non-private methods that update properties of the class.

18. An apparatus for invoking an object, comprising:

receipt means for receiving, from a caller, a request to invoke an object;

examination means for examining an argument in the request;

determination means for determining whether the object is immutable if the argument is an object;

reference means for passing an argument to the caller that includes a reference to the object and does not include a copy of the object; and

if the object is mutable, the reference means for passing a copy of the object to the caller.

19. The apparatus of claim 18, wherein the determination means comprises:

means for determining whether an immutability flag that is inserted into a class that describes the object is set.

21. A computer program product, in a computer readable medium, for loading a class, comprising:

instructions for loading a class;

instructions for inserting an immutability flag into the class;

instructions for determining whether the class is immutable;

instructions for setting the immutability flag if the class is immutable;

instructions for receiving a request to invoke a server application;

instructions for examining an argument in the request;

if the argument is an object, instructions for identifying the class that describes the object;

instructions for determining whether an immutability flag that is inserted into the class that describes the object is set, wherein the object is immutable if the immutability flag inserted into the class that describes the object is set;

if the object is immutable, instructions for passing an argument to the server application that includes a reference to the object, the argument not including a copy of the object; and

if the object is mutable, instructions for passing a copy of the object to the server application.

22. A computer program product, in a computer readable medium, for invoking an object, comprising:

instructions for receiving, from a caller, a request to invoke an object;

instructions for examining an argument in the request;

instructions for determining whether the object is immutable if the argument is an object; and

instructions for passing an argument to the caller that includes a reference to the object and does not include a copy of the object if the object is immutable; and

instructions for, if the object is mutable, passing a copy of the object to the caller.

EVIDENCE APPENDIX

The following three documents are submitted as evidence: Definition of *immutable* from Merriam-Webster Online Dictionary (2 pages), discussion of Immutable Objects from Java Practices (2 pages), and Immutable Objects in Java from Java Practices (3 pages).

Merriam-Webster OnLine

Merriam-Webster FOR KIDS Encyclopædia BRITANNICA
Merriam-Webster ONLINE Merriam-Webster COLLEGIATE® Merriam-Webster UN

HOME
PREMIUM SERVICES
M-WCollegiate.com
M-WUnabridged.com
Britannica.com
Multi-User Licenses
DOWNLOADS
WORD OF THE DAY
WORD GAMES
WORD FOR THE WISE
ONLINE STORE
HELP

Merriam-Webster Online Dictionary

Thesaurus

immutable

One entry found for **immutable**.

Main Entry: **im·mu·ta·ble**
Pronunciation: (ª)i(m)-'myü-t&-b&l
Function: *adjective*
Etymology: Middle English, from Latin *immutabilis,* from *in-* + *mutabilis* mutable
: not capable of or susceptible to change
- **im·mu·ta·bil·i·ty** /-ˌmyü-t&-'bi-l&-tE/ *noun*
- **im·mu·ta·ble·ness** /-'myü-t&-b&l-n&s/ *noun*
- **im·mu·ta·bly** /-blE/ *adverb*

For More information on "Immutable" go to Britannica.com
Get the Top 10 Search Results for "Immutable"

Merriam-Webste
◉ Dictionary
○ Thesauru

Browse by letter
A B C D E F G H I
N O P Q R S T U V

Browse words n
Immutable

Britan
Immutable

Pronunciation Symbols

Products Premium Services Company Info Contact Us Advertising Info Privacy P

Immutable objects

Discussion:

Immutable objects are simply objects whose state (the object's data) does not change after construction.

Immutable objects greatly simplify your program, since they

- are simple to construct, test, and use
- are automatically thread-safe and have no synchronization issues
- do not need a copy constructor
- do not need an implementation of `clone`
- do not need to be copied defensively when used as a field
- make good `Map` keys and `Set` elements (these objects must not change state while in the collection)
- the class invariant is established once upon construction, and never needs to be checked again

Make a class immutable by following these guidelines :

- always construct an object completely, instead of using a no-argument constructor combined with subsequent calls to `setXXX` methods
- do not provide any methods which can change the state of the object in any way - not just `setXXX` methods, but any method which can change state
- ensure no methods can be overridden - make the class `final`, or use static factories and keep constructors private
- make fields `final`
- if the state of a mutable object field is "owned" by the native class, and the intention is to allow *direct* access to the field only from within that native class, then, when the field "crosses the interface" (as in a `get` or `set` method, or in the constructor itself), then a defensive copy *must* be made, in order to maintain encapsulation.
- if the state of a mutable object field is not "owned" by the native class, then a defensive copy of the object field is not necessary

In *Effective Java*, Joshua Bloch makes this recommendation:

"Classes should be immutable unless there's a very good reason to make them mutable....If a class cannot be made immutable, you should still limit its mutability as much as possible."

Example

```
import java.util.Date;
```

```
/**
* Planet is an immutable class, since there is no way to change
* its state after construction.
*
* @is.Immutable
*/
public final class Planet {

  public Planet (double aMass, String aName, Date aDateOfDiscovery) {
     fMass = aMass;
     fName = aName;
     //make a private copy of aDateOfDiscovery
     //this is the only way to keep the fDateOfDiscovery
     //field private, and shields this class from any changes
     //to the original aDateOfDiscovery object
     fDateOfDiscovery = new Date(aDateOfDiscovery.getTime());
  }

  //gets but no sets, and no methods which change state

  public double getMass() {
    return fMass;
  }

  public String getName() {
    return fName;
  }

  /**
  * Returns a defensive copy of the field.
  * The caller of this method can do anything they want with the
  * returned Date object, without affecting the internals of this
  * class in any way.
  */
  public Date getDateOfDiscovery() {
    return new Date(fDateOfDiscovery.getTime());
  }

  // PRIVATE //

  /**
  * Primitive data is always immutable.
  */
  private final double fMass;

  /**
  * An immutable object field. (String objects never change state.)
  */
  private final String fName;

  /**
  * A mutable object field. In this case, the state of this mutable field
  * is to be changed only by this class. (In other cases, it makes perfect
  * sense to allow the state of a field to be changed outside the native
  * class; this is the case when a field acts as a "pointer" to an object
  * created elsewhere.)
  */
  private final Date fDateOfDiscovery;
}
```

Note that javadoc 1.4 includes the `-tag` option, whereby simple custom tags may be defined. One might define an `@is.Immutable` tag, for example, to document a class as being immutable.

See Also:

Defensive copying
Use final liberally
Factory methods
Validate state with class invariants
Document thread safety
Avoid JavaBeans style of construction
Model Objects

Would you use this technique?

Yes ○ No ○ Undecided ○ Vote

Add your comment to this Topic:

Comment:

Submit

Over 51,500 unique IPs (and 103,000 sessions) last month.

- In Memoriam : Bill Dirani -

ChurchillObjects.com

Immutable Objects In Java

Contents

The term *immutable* simply means unchangeable in object-oriented parlance. Immutable objects do not change once their constructor has executed. You may already know that the **java.lang.String** class is immutable, despite the appearance that it is not due to its convenient operator overloading. You may also know that if you need to write code that constructs a single **String** object over a number of separate statements, that it is more efficient to use the **StringBuffer** object. If this is not the case, don't worry about it. This article was writte to thoroughly explain immutable objects, how and why they work in the String and **StringBuffer**, and the best way for you to implement such a pattern in you own systems to make them more stable and robust.

Rather than being an interface or keyword that tells the JVM to enforce a rule, immutability is implemented as a design pattern that the programmer builds int the class. There are some general characteristics that are usually found in all intentionally immutable objects. These are:

- A class is instantiated and all of its properties set through the constructor (s). When the arguments are passed at creation, it eliminates any need for setter methods on the class that might be called later.
- The class is declared **final**. This prevents another developer from extending your class and implementing code that supports mutability.
- The properties of the class are **private**. This prevents other objects from modifying the attributes directly, and adds another layer of protection fron subclassed code over making the class **final**.
- Other than the constructor(s), there is no code in the class that changes th properties of the object. Any methods or events for the class must treat the private attributes as read-only. In this way, the final state of the object is determined when it is constructed, and no opportunity exists to change it for its lifetime.
- If method are implemented that perform operations on the data abstracted by the class, the result of those operations are another instance of the class containing the modified data. This is the case with **String** methods such a **trim()** and **toUpperCase()**.
- If the class is sufficiently general purpose in scope, it may be packaged wit a companion object that is mutable. This is the case with the **String** and

StringBuffer classes.

I will clarify these characteristics in more detail with code examples as we go along.

◀ Front Page

Reasons behind Immutability ▶

RELATED PROCEEDINGS APPENDIX

There are no related proceedings.